

Seismographic Data Compression

--Applying Modified Tunstall Coding--

Shu-Fang Newman
Computing and Software Systems
Institute of Technology
University of Washington, Tacoma
Date: December 12, 2006

Committee:

Edwin Hong, Ph. D., UWT Computing Software and Systems
Moshe Rosenfeld, Ph. D., UWT Computing Software and Systems

Seismographic Data Compression

Table of Contents

Seismographic Data Compression.....	1
--Applying Modified Tunstall Coding--	1
Table of Contents.....	2
Abstract	3
1. Introduction.....	4
A. Overview	4
Figure 0. Project Overview	5
B. Data Compression	6
C. Seismographic Data.....	7
2. SEED Format	8
3. Acquiring and Selecting the Seismic Data.....	9
4. Data Description Language of SEED	10
5. Tunstall Coding.....	12
Figure 1. Prefix-free Tree.....	14
Figure 2. Tunstall Prefix-free Tree	16
Table 1. Output Codeword Table Corresponding to the Tunstall Tree	17
6. Encoding Modified Tunstall	17
Figure 3. Design of Modified Tunstall Coding.....	18
Table 2. Partition Differences into Ranges	20
7. Decoding Modified Tunstall	22
8. Why the Design of Modified Tunstall	23
9. Compression Ratio of Modified Tunstall Coding.....	26
Table 3. Compression Ratios of Modified Tunstall	28
10. Comparing Modified Tunstall with Steim1 and Steim2	29
Table 4. Comparison of Steim1/Steim2 and Modified Tunstall	32
Table 5. Comparison of Steim1, 2 and MTC at DGAR/PTCN Stations	33
11. Comparing Modified Tunstall with Linear Prediction Coding.....	34
Table 6. Comparison of LPC and Modified Tunstall at DGAR/PTCN Station.....	36
12. Rewriting Modified Tunstall Coding in DDL	37
Figure 4. View of the Dictionary Header.....	37
Figure 5. Snapshot of an Example of the Dictionary Header	39
Figure 6. View of the ADCH of Figure 5	40
Figure 7. View of key 3	41
Table 7. SEED Compression Ratios of Steim and MTC	42
13. Conclusion	42
14. References	43
Appendix I: Application of Predictcode.m	44
Appendix II: The List of Selected File Names	45
Appendix III: Seismic Data Request Form.....	46

Abstract

The Standard for the Exchange of Earthquake Data (SEED) is a commonly used file format in the seismology field. Steim1 and Steim2 compression schemes, i.e. lossless data compressions, are used in SEED format and are written in Data Description Language (DDL), which has computational limitations making it difficult to implement many standard compression algorithms. Steim1 and Steim2 are fixed compression methods, which assign each incoming data sample to fewer bits than 32-bit, regardless of the essence of the data. This project modified the Tunstall compression scheme to gain a better compression ratio of seismic data and rewrote the compressed data in the DDL of SEED format file. This project pre-computed the statistics on seismic profile bases and, accordingly, wrote the Modified Tunstall data compression description. This strategy improves compression ratios over Steim1 and Steim2. The average compression ratio of the Modified Tunstall Coding in this project was 3.18 when the length of the output codeword was fixed at 10 or 11 bits. The Modified Tunstall Coding showed better compression than Steim1 by an average of 30.78% and Steim2 by an average of 5.16%. When comparing the Modified Tunstall Coding with the Linear Prediction Coding, which is not possible to implement in DDL, the Modified Tunstall Coding was only 7.95% worse on average.

1. Introduction

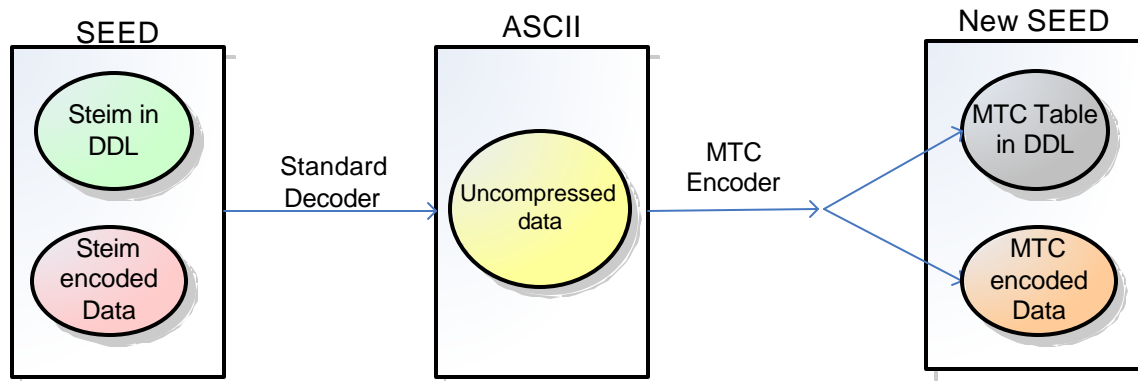
Progress in computer technology has inspired and changed human life. In the last decade, especially with the growth of the Internet, humans have gradually and principally relied upon storing and transmitting information in electronic ways. Information represented by bytes has gradually become more extensively used in a digital form. Consequently, people can now gather information easily and quickly. Computer technology's impact on humans and society is affecting and transforming life more than we realize. However, it requires large electronic depositories. A prerequisite for this mandatory memory storage is knowing how to store this information. One of the best ways to resolve the escalating need for memory storage is to represent information within a compact size. The Internet and networking are now commonly used far and wide, yet the digital informational format is still limited by transmission speed.

A. Overview

This project applied the Tunstall compression scheme by observing the statistics of the seismic data, which is a sequence of integers, and then, using the statistics, wrote Modified Tunstall Coding. The compressed data was written in Data Description Language (DDL), which is part of the Standard for the Exchange of Earthquake Data (SEED) format (see Figure 0). The creation of Modified Tunstall Coding in DDL is intended for the convenient of SEED users who may not know what underlies SEED or DDL. The intention of this project was to improve the compression ratios of Steim1 and Steim2 compression schemes. The improvement resulted in a modification of Tunstall Coding into DDL of SEED.

The description of this project includes three parts: 1. Introducing the background of data compression and the innate of the seismic data; and then focusing attention on DDL of SEED; 2. Illustrating the design of the modified compression scheme based on the limitation of DDL; and 3. Measuring and comparing, through experimentation, the compression performance, and writing the new compression scheme in DDL.

Figure 0. Project Overview



A brief introduction starts with data compression and seismic data. It continues with illustrations of acquiring and selecting seismic data to obtain an understanding of the essence of the seismic data. Then the project focuses on the DDL of SEED format. A description of these endeavors constitutes the first part of this paper: “1. Introduction,” “2. SEED Format,” “3. Acquiring and Selecting the Seismic Data,” and “4. The Data Description Language of SEED.”

SEED format is widely used in the seismic field. To be supported in SEED, the data format must be expressible in SEED DDL. How would this project benefit the large populations of SEED users? This question led to the fine tuning of this project. This led to other questions: What can the DDL language of SEED do? How can this project efficiently make the compression scheme of DDL represent the data? These inquiries guided the design discussed in the second part of this project.

The compression scheme expressed in DDL requires simple computations to fit the limitations of DDL. Tunstall coding was found to fit DDL. However, applying Tunstall Coding may result in the use of a large size of the Codeword Table, which needs to be stored as part of the compression and results in of a poor compression ratio. Therefore, the deficiency of Tunstall Coding motivated this project to modify the existing coding to fit the design criteria of this project. This paper details the encoding and decoding of Modified Tunstall Coding, as well as why the Modified Coding was designed by analyzing the seismic data. The second part of the paper comprises four

sections: “5. Tunstall Coding,” “6. Encoding Modified Tunstall,” “7. Decoding Modified Tunstall,” and “8. Why the Design of Modified Tunstall.”

Finally, the last part of this paper addresses the comparison of the compression ratio of Modified Tunstall coding, the Linear Prediction Coding, and Steim1 and Steim2. The last Section of this project embedded Modified Tunstall Coding in DDL to make it convenient for SEED users who may not know what underlies SEED or DDL. This part of the paper collects four Sections: “9. Compression Ratio of Modified Tunstall Coding,” “10. Comparing Modified Tunstall with Linear Prediction Coding,” “11. Comparing Modified Tunstall with Steim1 and Steim2,” and “12. Writing Modified Tunstall in DDL.”

B. Data Compression

Data compression is often referred to as coding. The primary objective of coding is to minimize the amount of data transmission and/or to conserve storage space. Most source messages, such as text, image, and video, are naturally redundant. Normally, the desired information is expressed in excess data amounts of the actual need. Data compression allows devices to perform (process) original data in reduced bits by identifying the data’s structure. Data compression, currently, is mainly classified as lossless or lossy schemes. Lossless data compression retrieves information in the same form as the original; while, lossy involves some loss of data [\[1\]](#). When the original data exactly equals the decompressed data, the process is called lossless compression, while lossy compression shows the original data do not equal the decompressed data [\[2\]](#). One of the criteria to evaluate the performance of a compression scheme is to examine the compression ratio. How to measure the compression ratio will be introduced in a later section of this paper. Retrieving lossless data is required in certain fields, such as medical research, geophysics, and telemetry data communication. Seismographs produce integer values that have unique characteristics. These characteristics differentiate seismographic (seismic) data from the still images and video data that are the focus of most lossy compression efforts. Seismographic data mainly requires lossless compression. This project only involves in the lossless compression scheme.

C. Seismographic Data

A seismograph measures ground motion produced by earthquakes, volcanoes, and other sources. Ground motions typically last from several tens of seconds to many minutes, depending on the size of the earthquake and the sensitivity of the seismograph. Seismography records and measures simply-processed or unprocessed earth motion data. A seismograph station processes data twenty-four hours per day, seven days per week, and then reports it to anywhere in the world. A seismograph laboratory simulates seismograms that are routinely recorded during any 24-hour period of an earthquake laboratory is functioning. The height of the recorded waves on the seismogram (wave amplitude) greatly magnifies representation of the actual ground motion. A seismograph produces seismogram records which report the ground motions over the passage of time. The horizontal axis of a seismogram represents the time in seconds, and the vertical axis corresponds to the ground displacement in millimeters. A straight line in a seismogram, called "noise," means no ground motion reading. Data from a seismograph registers not only earthquakes; it may include other noise, such as rock falls, ice quakes, or electrical noise from telephone lines. Such noise or vibration is usually easy to distinguish from an earthquake.

A recording of an earthquake has recognizable characteristics. Typically, one can recognize the arrival of different wave types: Primary waves (P-wave), the fastest traveling waves; Secondary waves (S-wave), shear waves; and Surface waves.

Seismographic data is composed of signed 32-bit integers. Each data sample consists of four bytes binary storage under normal, non-compression conditions. Without compression, a seismic station with 3 components recoding at 20 samples per second, exceeds 20 megabits per day. A network with 100 stations could produce 2 gigabits per day. However, an analysis of the characters of the seismic data may allow the data to be represented by fewer bits. In general, a seismogram shows that seismic data are highly related within a time series. They are highly predictable in a sequence of consecutive samples if the previous few samples are known.

Seismic data are transferred from a station processor to a data collection center, then to a data management center, and, finally, to an end user. One of the most important organizations, the Data Management Center (DMC) at Incorporated Research Institutions

for Seismology (IRIS), collects seismic data “from a variety of Data Collection Centers and is responsible for the long term archive and distribution of all IRIS generated data” [3].

2. SEED Format

The DMC of IRIS has adopted the Standard for the Exchange of Earthquake Data (SEED) [4], which was defined by the Federation of Digital Seismographic Networks (FDSN) to help earthquake research communities to exchange, between institutions, unprocessed digital seismologic data. SEED format became an international standard for the exchange of digital seismic data. The majority of seismic data, stored in or archived from DMC, are in FDSN SEED format. SEED is the principal format for the DMC datasets. Seismologists use SEED format to transmit seismic data by electronic means, such as a packet switching network. SEED includes Steim1 and Steim2 data compression schemes. Part of SEED includes a language, Data Description Language (DDL), which parses and disassembles native data format. DDL commands are part of SEED format. Large populations of the seismologic field depend on DDL, which is embedded in SEED format, for their research on seismic data. If this project provides exactly the necessary DDL commands as a part of a SEED file, then SEED’s users do not need to change the way they read the data; the users can still read the data. The intent is to make the process convenient for the large populations and the SEED users who do not need to know DDL language nor understand how DDL works. Consequently, this project’s intent was to design a new compression scheme, embedded in DDL language, which describes how to read this data.

The IRIS SEED reading program (RDSEED) [5] was developed to help researchers convert their datasets into trace formats for which analysis tools already existed. RDSEED is a commonly used tool that can extract many different types of data, such as AH, CSS, and SAC ASCII, from a FDSN SEED format volume. This project requested seismic data from the DMC of IRIS and extracted the data by applying RDSEED. The target of this project was to create an efficient data compression scheme and to replace the Steim1 or Steim2 compression schemes of DDL. Once the new scheme

was expressible in DDL and replaced Steim1 or Steim2 of DDL, then the new DDL would be able to decode data by SEED reading programs. This new scheme would enable users of RDSEED to read data compressed using the new SEED format, which would benefit all RDSEED users by conserving their storage space and reducing time to transform data and/or computation time. The details of DDL are illustrated in the Section “4. Data Description Language of SEED.”

3. Acquiring and Selecting the Seismic Data

One of the major tasks to perform a better data compression scheme is by identifying the data’s structure. To do this and prior to acquiring seismographic data, it is best to understand and facilitate what type of seismographic data is available. A seismographic networking organization may operate a heterogeneous mix of seismometer types to monitor a variety of types of earthquake activities. A seismometer may measure signals from 10^{-3} samples per second (Hz) to 80 Hz; it depends upon a variety of usage purposes for seismographic data. The Broad Band type of data was chosen for this project because it commonly applies to the field of seismology and usually includes a large range of amplitudes. A Broad Band sensor records waveforms from regional earthquakes and teleseismic events for research purposes. The Broad Band type of data by the DMC of IRIS is archived to most frequencies from 10 Hz to 80 Hz. The Broad Band files are identified by suffixes: BHZ, HHZ, BHE, HHE, BHN, or HHN.

This project selected the Broad Band channel data from the DMC on four stations: two stations in the II Network, DGAR (Diego Garcia, Chagos Islands, India Ocean) and PTCN (Pitcairn Island, South Pacific), and another stations, ANMO (Albuquerque, New Mexico, USA), and MAJO (Matsushiro, Japan) in the IU Network. The chosen starting time was 00 hours 00 minutes 00 seconds on January 1, 2006 and ended at 00 hours 00 minutes 00 seconds on January 2, 2006. (See [Appendix III](#) for the actual Seismic Data request form.)

The requested files, from the DMC, of this project are in SEED format. One example of the WebRequest form for requesting data from DMC of IRIS is listed in Appendix II at <http://www.iris.edu/data/WebRequest.htm>.

Files formatted in SEED are usually compressed either in Steim1 or Steim2 of DDL. To read SEED files for the further analytical purposes, this project ran all the files in RDSEED 4.6 on a Unix computer specifying the parameter of the d options (retrieve all selected data, selection by list of record numbers), the parameter of outputting the SAC_ASC (SAC ASCII) format, and leaving other parameters as default. After running the decompression files using RDSEED, the output files for the particular seismogram were named by the first recoded data of the beginning time, station and component (see [Appendix II](#)) as yyyy.ddd.hh.mm.ss.ffff.SSSSS.CCC, where yyyy is the year, ddd is the Julian day, hh.mm.ss.ffff is the starting recorded time of the day, SSSSS is the station name, and CCC is the component name. For example: A SAC ASCII data output format file named 2006.031.13.30.20.0104.IU.ANMO.10.BH1.Q.SAC_ASC reveals the data is produced beginning from the year of 2006 at 13 hours 30 minutes 20:0104 seconds of the 31st Julian day at the location 01 of the BH1 component of the station ANMO of the IU network. The recovered files from a SEED-formatted file may be decompressed into more than one file by RDSEED. The number of file depends on the initial recoding time and the given components of the station. The recovered file includes data headers and data records. Seismic data records are a sequence of integers.

All the chosen channels in this project were BHE, BHZ, BHN, BH1, and BH2 Broad Band files. The location in each channel might be recorded as a 00, 01 or 10 location. 01 and 10 location involved 40 samples per second (Hz), and 00 location involved 20 Hz.

4. Data Description Language of SEED

The actual seismic data in DDL language is represented as a sequence of integer differences. If users correctly interpret DDL, the final output will be a sequence of code which will turn out to be integer differences. DDL is a language, but part of SEED format can include DDL. To be supported in SEED, the data format must be expressible in

SEED DDL. Writing data in DDL is one part of SEED format. Readers can write data in Data Description language, and then the language itself will describe how to decode the data. SEED format includes DDL.

One of the advantages of using DDL is intending to be convenient for users. DDL commands are part of SEED format. The users of DDL do not need to change the way they read the SEED data, and they can still use the new format and process the data in their original way. This project's intent was to provide the necessary DDL commands as part of the SEED file. In addition to the majority of the researchers widely adapting SEED format, this project aimed to develop the new compression scheme, which replaces Steim1 and Steim2 in DDL and executes the decompression job for SEED. The SEED users may not know the underlying of SEED format, nor DDL, but the users would gain a better compression ratio. In order to be supported in SEED, the experimented-upon files of this project were written and replaced in DDL of SEED.

DDL supports several different data families, such as integer, integer differences compression, and text. Any compression scheme expressed in DDL can be read by any SEED reader. This paper only concentrates on the related coding, which is defined in DDL as Family 50: integer differences compression. However, the language is a simple assembling language. DDL cannot perform conditional looping for decoding data. It supports only fundamental operations to resemble the Steim1 and Steim2 Compression Algorithm in Family 50. To fit the limitations of the language, a thorough study of DDL is critical.

Family 50 of DDL, integer differences compression, is basically designed to decode Steim1 and Steim2 compression algorithms. The Family is composed of several records called keys. Keys describe how to decompress the seismic data. Two keys, key 1 and key 2, are specially defined in the Family 50. A number of control type keys, after the first two keys to a specified key, are designed to interpret the compressed data by all possible compression key values (control codes) derived in key 2. Each key has a unique number to provide different control code. Keys are separated by tildes, ~.

The first key, key 1, of the integer differences compression instructs to access the forward integration constant (the first piece of the data) and the reverse integration constant (the last piece of the data). Key 2 of the Family 50 provides information on how

to interpret the control codes and how to carry out the request of the control codes. Key 3 to the rest of the keys excludes the last key and describes how to decompress data according to the compression keys' value. The last key is an optional key. It is used for cleaning up operations or actions that have to be done at the end of reading a block. This project did not use the last key.

Each key of Family 50, excluding the first two keys, starts with an upper-case alphabet (field) which is followed by the value of the control code. Each key provides a couple of fields. A field in keys contains the actual parser information for operations to carry out. Fields contain commands about how to interpret the actual data and how to instruct and decode them. Each field has a single upper-case alphabet symbol followed by numeric parameters separated by a comma. Fields within a specified key are divided by one space.

DDL does not support many operations (primitives) in the fields. Especially, DDL can not perform conditional looping for decoding data. This constraint makes it impossible to implement many of the standard compression techniques in DDL, which involve more complicated computations. The fields of DDL only contain operations, such as copying/recording primitives, extraction primitives, sign primitives, and a few other primitives. The copying/recording primitives copy bytes/bits from the input data stream into a working buffer, while the extraction primitives extract specified bytes/bits from the working buffer as numbers by applying offset and scale factors. DDL also includes primitives of repeating successive fields and discards the results or the contents of the working buffer.

The studying of DDL language helped this project to understand the following fact: The design's criterion of the compression scheme expressed in DDL requires simple computations to fit the limitations of DDL. The next section introduces Tunstall Coding, and the sixth section explains why this project modified Tunstall Coding to fit DDL.

5. Tunstall Coding

Tunstall Coding provides the lossless data compression scheme. A natural way to compress the sequence of symbols is to assign more frequent symbols to fewer codeword

bits and less frequent symbols to more codeword bits. Tunstall Coding takes the natural idea in a different way, which makes Tunstall Coding unique in the following way. Tunstall Coding uses the natural idea in reverse order: assign a set of input strings, which are about the same frequency, to fixed-length codeword bits. The set of input strings is a prefix-free set of varying length strings.

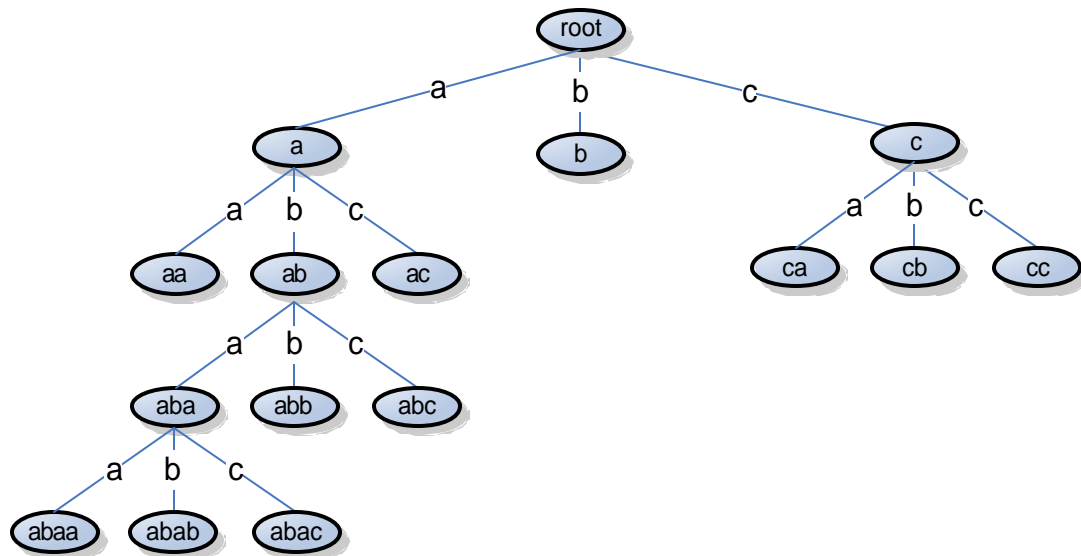
It is important to be a prefix-free set to ensure unique encodeability on a sequence of symbols that will be encoded. “Encodeability” means to take any sequence of symbols and to divide them uniquely into input strings. “Unique encodeability” means that the project is able to encode any sequence of symbol into input string in only one way. Readers may note that “a sequence of symbols” is not the codewords. This project intent was to have a set of strings that could be used to encode any sequence of symbols, or any sequence of symbols that are able to uniquely divide into input codewords (input strings) from the set.

All output codewords of Tunstall Coding have an equivalent length. One advantage of the variable-to-fixed length code is error resilience. Unlike the fixed-to-variable length codes, a single bit flip cannot destroy the variable-to-fixed-length code. The errors in fixed-length codewords do not propagate; a single bit flip only introduces one error in the output.

A convenient way to represent a prefix-free set of input strings is to create a tree. To construct the prefix-free tree, begin from a single node (the root node) and then add branches (edges) for each different symbol. Each edge corresponds to a symbol (different symbol). These branches end in leaf nodes. Each leaf node of the tree represents an input string which is composed from the symbols. To add branches and leaf nodes, repeat this instruction to expand the tree to the desired number of leaves by exploring specific leaf nodes, one at a time. Once expanded, the leaf node changes to an internal node. Be aware that no input string is terminated at any internal node in the prefix-free tree. The input string can be obtained by traversing the tree from the root to the internal node(s) and to the desired leaf nodes; write respectively each corresponding symbol of the branches to compose the input string. All the leaves of the tree represent the prefix-free set of input strings. An example of the prefix-free tree is shown in Figure 1, where each leaf node represents a prefix-free input string.

A prefix-free tree can be built to help the Tunstall algorithm to represent the map from sequence of symbols to the prefix-free set of input strings. The mapping of the prefix-free Tunstall tree and the output Codeword Table will be illustrated in the last paragraph of this section; meanwhile, the concept of drawing the prefix-free Tunstall tree will be described in the following paragraph.

Figure 1. Prefix-free Tree



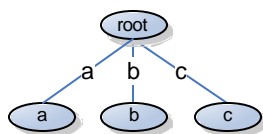
First, let m be the initial alphabet size of the sequence of symbols, which is desired to be compressed. Let the length of the targeted output codewords be n , where $2^n > m$. (Note: Carefully choose the length n : If n is too big then the compression result may be poor; if n is too small, then the size of the codeword table may not be large enough to represent all the symbols.) Calculate the probability of each symbol. Form a prefix-free tree, as previously instructed, beginning with a root and m leaves, where each edge is labeled with the m alphabet size and each leaf node represents the input string. The probability of the input string is the occurrence of its associated symbols. While the total number of leaves of the tree is less than or equal to $2^n - m$, expand the highest probability of the leaf node, which is the highest probability of all of the current leaves, to have m children, where the new edges are labeled with the m symbols and each new input codeword increases one symbol more than its parents. Stop when the number of

leaves is greater than $2^n - m$. The probability at each leaf node is the probability of the occurrence of its associated symbols. (Note: The next desired expanded leaf is chosen from the highest probability of all of the current leaves; the probabilities of the m new leaf nodes equal each probability of their newly labeled edges times the previously chosen highest probability.) Each path from the root to a leaf node produces the probability of an input string which is the product of each edge's probability. Each input string on the tree will map to one unique codeword.

An input string can be obtained by traversing the tree from the root to the internal node(s) and to the desired leaf nodes. The collection of the input string introduces a prefix-free set of input strings. The Codeword Table maps from input strings to output codewords which have n -bit fixed length. Applying the Codeword Table transforms the sequence of symbols into an output codeword to compress the desired data. For decodeability, compressed output requires more than the codewords: The output comprises the Codeword Table and the transformed n -bit length codewords.

The following example explains and clarifies the details of building the prefix-free tree and the Codeword Table. Let the desired compressed sequence of symbols contain only three symbols a , b and c , where the probabilities are $p(a) = 0.1$, $p(b) = 0.4$ and $p(c) = 0.5$. Form a tree from the three symbols. Let the target length of the codeword be 4-bit, and then expand the tree until the leaves reach, but not exceed $2^4 - 3$. The first level of the tree has three edges a , b and c which respectively follows their own codeword which resides on each leaf.

$p(a) = 0.1$
 $p(b) = 0.4$
 $p(c) = 0.5$



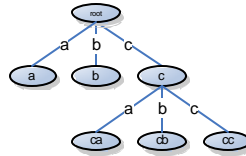
Since $p(c)$ is greater than $p(b)$ and $p(a)$, explore the leaf with the highest probability $p(c)$ and begin to expand the c leaf to have three more children ca , cb and cc . The probabilities of these leaves are obtained from the highest probability $p(c)$ times the new edges' probabilities $p(c)$, $p(b)$, and $p(a)$.

Each leaf node of the second level of the tree

$$p(ca) = p(c) * p(a) = 0.5 * 0.1 = 0.05$$

$$p(cb) = p(c) * p(b) = 0.5 * 0.4 = 0.20$$

$$p(cc) = p(c) * p(c) = 0.5 * 0.5 = 0.25$$



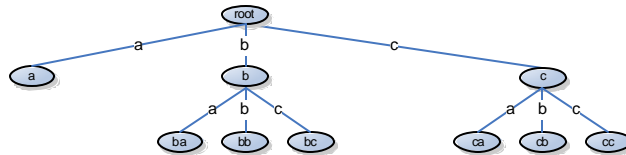
Choose the b leaf and follow the expanding procedure, since $p(b) > p(cc) > p(cb) > p(a) > p(ca)$. The probabilities of the new leaves are calculated as follows.

The new leaf nodes

$$p(ba) = p(b) * p(a) = 0.4 * 0.1 = 0.04$$

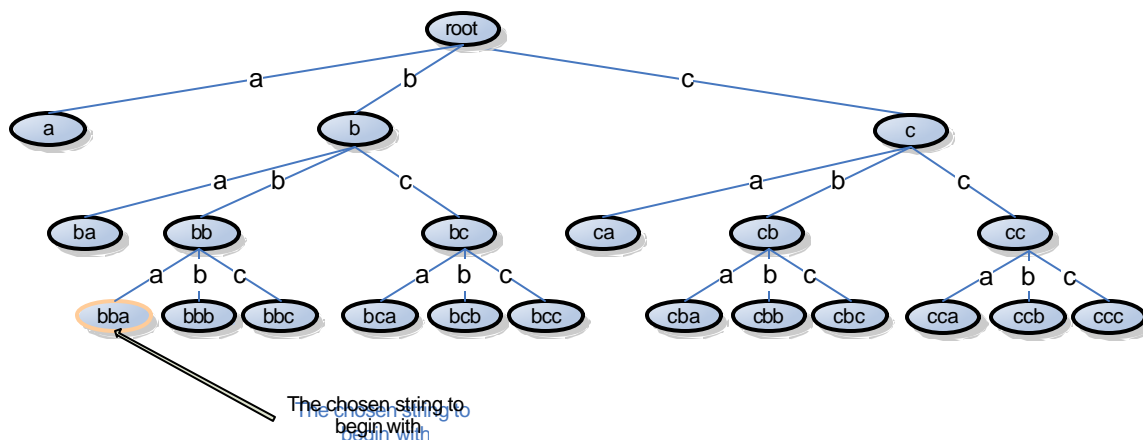
$$p(bb) = p(b) * p(b) = 0.4 * 0.4 = 0.16$$

$$p(bc) = p(b) * p(c) = 0.4 * 0.5 = 0.20$$



In this stage, the tree has only 7 leaves which are less than $2^4 - 3 = 13$; the expanding work needs to be continued. The next chosen leaf is the cc string because $p(cc) > p(bc) > p(cb) > p(bb) > p(a) > p(ba)$. (Note: The nodes b and c, at the first level of the tree with probability 0.5 and 0.4, are no longer identified as leaves; instead, they become internal nodes.) The input string only terminates at the leaves instead of at the internal nodes. Because the number of leaves are less than or equal to 13, apply the same processes to expand the highest probability over all, where 13 is obtained from $2^4 - 3$. Stop to expanding the tree when the number of leaves reaches 15, where $15 > 2^4 - 3$, as Figure 2 shows.

Figure 2. Tunstall Prefix-free Tree



To make the Codeword Table, begin by choosing an arbitrary leaf from the Tunstall tree. Example: Begin with the bba string, which traverses from the root through the internal nodes b and bb to the bba leaf (see Figure 2). Let the first explored output codeword be 0000 in binary. The input string bba corresponds to 0000 in the Output Codeword Table. (Notice: The output codeword, 0000, has 4-bit length, which was defined initially in the beginning of this paragraph.) Follow the same procedure to complete the table by developing any undiscovered leaf and assign each input string a consecutive binary number (see Table 1).

Table 1. Output Codeword Table Corresponding to the Tunstall Tree

Input string	bba	bbb	bbc	bca	bcb	bcc	cba	cbb
Output codeword	0000	0001	0010	0011	0100	0101	0110	0111
Input string	cbc	cca	ccb	ccc	a	ba	ca	
Output codeword	1000	1001	1010	1011	1100	1101	1110	

6. Encoding Modified Tunstall

The Modified Tunstall Coding¹ applied the Tunstall compression scheme by pre-computing the statistics on the profile bases of the seismic data. This section illustrates the basic design principles of Modified Tunstall Coding and gives an example to demonstrate the design. The effects of these pre-computational statistics of Modified Tunstall Coding are demonstrated on compression and shown in the section “10. Comparing Modified Tunstall with Steim1 and Steim2.”

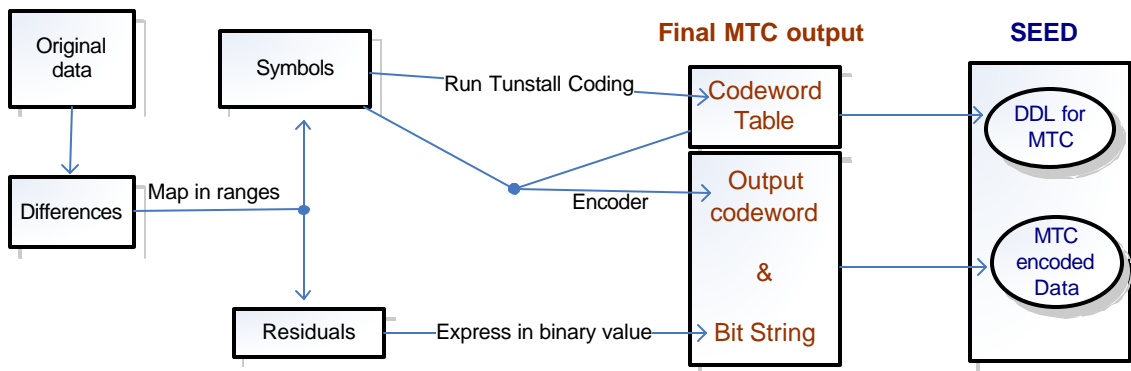
In order to encode a sequence of integers using Tunstall Coding, each different integer must be assigned to a unique symbol. It is impractical to represent every various signed 32-bit integer into an assorted symbol, which may require up to 2^{32} different symbols. The modification of Tunstall for a sequence of integers, especially for the seismic data, is necessary. This project mainly modified Tunstall Coding as one symbol

¹ Edwin Hong, PhD, of the University of Washington, Tacoma provided initial guidance toward the idea for this original work.

representing a range of integers. If all input strings in a range have the same probability of occurring, and the length of the range is a power of two then no further compression is possible (assuming a memoryless model). Tunstall Coding is an optimal variable-to-fixed length for a memoryless source and its achieved rate is close to the entropy. The ranges of Modified Tunstall Coding are designed to be to the power of two. Therefore, each integer's datum is represented as one representing symbol and one residual. The absolute value of the residual, a positive decimal number, is a number no larger than the size of the range. For negative differences, the residual is negative. (Note: The number in this paper, unless specified, is in decimal.) When a residual converts to a binary number, it is called a bit string. The conversions are shown in later paragraphs.) The length of the corresponding bit string is displayed in Table 2. Because the representing symbol is encoded, but the residual is not, each length of a bit string is allocated the power of two; it is expected that all lengths to the power of two will be used and that all of the different combinations of the bit strings represent the different possibilities.

To accommodate the limitations of DDL and to gain a better integer seismic data compression, this project modified Tunstall Coding in the following five processes (see Figure 3):

Figure 3. Design of Modified Tunstall Coding



- (1). Read each piece of input data sample, a sequence of integers, which is desired to be compressed. Calculate the difference between current input data and previous input data where

$$\text{the difference} = \text{current input data} - \text{previous input data.}$$

(2). Each difference is transformed to two parts of codewords: output codeword and output bit string. Design the mapping from a range of integers to a corresponding symbol and assign the length of the bit string to the symbol (see Table 2). (The reason for designing the set of ranges of Table 2 is illustrated in the last paragraph of the section “7. Acquiring and Selecting the Seismic Data.”) By applying Table 2, the reader can find the corresponding range wherein each difference falls.

Represent each difference as the found symbol. Calculate the residuals by subtracting the absolute value of the magnitude 2^r for positive differences or -2^r for negative differences, where r is the length of the corresponding bit string.

If the difference is positive, then the residual = value of positive difference $- 2^r$. (see Table 2)

If the difference is negative, then the residual = value of negative difference $+ 2^r$. (see Table 2)

(Note: The negative difference residuals are negative.)

Convert every absolute value of the residuals to a binary bit string where the length of the bit string is r . Respectively, store the pair symbol and binary sequence for further use.

A couple of examples for demonstrating the conversion of differences to the symbols with binary sequences are as follows: The number of difference, 3, falls into the range of 0 to 127 as defined by Table 2. Represent the difference, 3, by the symbol “a” where the magnitude of the “a’s” range is 0. Now the residual 3 comes from 3 minus 0. Convert the value of residual 3 to the binary bit string 0000011. (Notice: The length of corresponding bit string is 7 by the design of Table 2. Therefore, the binary sequence is coded in seven digits 0000011 instead of two digits 11.) Clearly at this stage, number 3 is represented as the pair (a, 0000011). Apply the same behaviors to convert differences 254 and -600 to be the pairs (symbol, bit strings).

$$\text{Ex.1. } 254 = 128 + 126 = 2^7 + 126$$

Convert 254 to => (b, 1111110(binary))

$$\text{Ex.2. } -600 = - (512 + 88) = - (2^9 + 88)$$

Convert -600 to => - (D + 001011000(binary)) = (D, 001011000(binary))

Converting a negative number to a symbol with binary sequences may seem a bit complicated in this project. But an equivalent way of converting them is to take the absolute value of the negative number to the pair (symbol, bit strings) and

change the lower-case symbol to an upper-case symbol. (Notice: If the converted symbol is a lower-case symbol, then the lower-case symbol implies the number of the difference is a non-negative number (see Ex.1) by the design of Table 2; otherwise, an upper-case symbol must correspond with a negative value (see Ex.2).) Consequently, the negative number -600 can be represented as (D, 001011000) pair without a negative sign because of the upper-case D. Later in the process of decoding, when an upper-case symbol is encountered, then the negative sign should be applied to the decoded number.

- (3). Calculate the probability of each symbol that is obtained from (2).
- (4). Run the Tunstall algorithm.
- (5). Transform the original data and yield the output codewords with their corresponding bit strings; these are recorded in pairs. Each pair consists of one codeword and a sequence of bit strings. (Note: If an output codeword is transformed from the leaf of the first level of the Tunstall tree, then there is only one single bit string in that pair; otherwise, the number of the bit strings depends on the number of levels.) These regulations are defined according to Table 2. The encoded output comprises two segments: the codeword table and the combination of the pairs (output codewords, bit strings).

Table 2 Partition Differences into Ranges

Ranges of Integers	Representing Symbol	Number of bits in Corresponding Bit string	Ranges of Integers	Representing Symbol	Number of bits in Corresponding Bit string
0 to $(2^7 - 1)$	a	7	-1 to $-(2^7 - 1)$	A	7
2^7 to $(2^8 - 1)$	b	7	-2^7 to $-(2^8 - 1)$	B	7
2^8 to $(2^9 - 1)$	c	8	-2^8 to $-(2^9 - 1)$	C	8
2^9 to $(2^{10} - 1)$	d	9	-2^9 to $-(2^{10} - 1)$	D	9
2^{10} to $(2^{11} - 1)$	e	10	-2^{10} to $-(2^{11} - 1)$	E	10
2^{11} to $(2^{12} - 1)$	f	11	-2^{11} to $-(2^{12} - 1)$	F	11
2^{12} to $(2^{13} - 1)$	g	12	-2^{12} to $-(2^{13} - 1)$	G	12
2^{13} to $(2^{14} - 1)$	h	13	-2^{13} to $-(2^{14} - 1)$	H	13
2^{14} to $(2^{15} - 1)$	i	14	-2^{14} to $-(2^{15} - 1)$	I	14
2^{15} to $(2^{16} - 1)$	j	15	-2^{15} to $-(2^{16} - 1)$	J	15
2^{16} to $(2^{17} - 1)$	k	16	-2^{16} to $-(2^{17} - 1)$	K	16
2^{17} to $(2^{18} - 1)$	l	17	-2^{17} to $-(2^{18} - 1)$	L	17

The following example clearly illustrates the processes of compressing a sequence of data by using the previously defined Modified Tunstall Coding and then producing a compressed data which includes a codeword table, codewords, and bit strings.

- (1). Read a sequence of data, which in this project is a sequence of integer seismic data,

8704, 8507, 8438, 8444...

and calculate the differences

$$8507 - 8704 = -197$$

$$8438 - 8507 = -69$$

$$8444 - 8438 = 6$$

(Notice: The differences are 8704, -197, -69, 6... instead of -197, -69, 6....

Because of the decodeability, one must copy the first datum 8704 as the first difference, rather than -197; without recording the first piece of data, it would be impossible to retrieve the original sequence of data.)

- (2). Look up Table 2 to transform the differences. The difference 8704 falls in the range of 8192 to 16383, and the representing symbol is “h” as shown in Table 2. Convert the residual, $8704 - 8192 = 512$, to the binary sequence as 000100000000.

$$8704 = 8192 + 512 = h + 000100000000(\text{binary}) \Rightarrow (h, 000100000000(\text{binary})).$$

The second difference, -197, falls in the range of -128 to -255 and the representing symbol is “B.”

$$-197 = -(128 - (-69)) = -(2^7 - (-69)) \Rightarrow (B, 1000101(\text{binary})).$$

The number of the difference, -197, is represented by the pair (B, 1000101). The third difference, -69, is represented by the pair of (A, 1000101). Number 6 is (a, 0000110) and so on. Subsequently, follow the previously Modified Tunstall explanations, found in preceding steps (3) and (4), to make a Tunstall tree by calculating probabilities of the symbols.

- (4). Using the last step, create the Codeword Table according to the Tunstall tree, where the size of the Table is not greater than $2^n - m$. (Note: Each alphabet symbol is assigned as an n-bit length codeword, see Table 1.)
- (5). Finally, transform the sequence of data and yield the output codewords with

bit strings. To be clear, the encoded output includes the Codeword Table described in step (4) and output codewords combined with bit strings.

7. Decoding Modified Tunstall

Decoding Modified Tunstall requires the Codeword Table and the output codewords combined with bit strings. Initially, read one n-bit length codeword at a time, where n is the default length of the codeword, look up the range of integer of Codeword Table, and then find the representing symbol. Take one representing symbol of the input string at a time and convert it to the absolute value of the magnitude of the matching range by the following conditions:

(0). Define the Modified ASCII Value as

Modified ASCII Value = ASCII value of the symbol – 97 (for lower-case character) ... (1)

or Modified ASCII Value = ASCII value of the symbol – 65 (for upper-case character) ... (2)

(1). If the result of the ASCII value of this symbol minus the ASCII value of symbol a, 97, is a non-negative number, then the symbol must be a lower-case character which means the Modified ASCII Value is a non-negative number (see formula (1)).

(2). Otherwise, the symbol must be an upper-case character which determines that the Modified ASCII Value is a negative number (see formula (2)).

(Keep in mind: After retrieving the upper-case ASCII value of the absolute value of the magnitude matching range, a negative sign is required to perform the retrieving value, which will be negative.)

Recall that the length of the bit string in Table 2 starts at the basic length 7. Therefore, for the first step, retrieve an output codeword as the input string; then locate the symbol(s) of the input string to the absolute value of the magnitude matching range(s). Because the basic length of bit string is 7, therefore 6 (7 minus 1) has to be added to Modified ASCII Value (see formula (1) and (2)), if the ASCII value of the symbol does not equal 0; otherwise, the Modified ASCII Value is 7. The Modified ASCII Value in this stage is not the final difference data yet because the bit string binary code needs to be transformed into a decimal number.

```

if      lower-case Modified ASCII Value = 0 return  bit string in decimal
else if upper-case Modified ASCII Value = 0 return  -bit string in decimal
else if lower-case Modified ASCII Value > 0
                                return  bit string in decimal + 2Modified ASCII Value
else
                                return  -(bit string in decimal + 2Modified ASCII Value)

```

(Be aware: An upper-case character represents a negative number by the design of Modified Tunstall Coding. Therefore, the returning value needs to be modified as a negative number by adding a negative sign for the entire number

(bit string in decimal + 2^{value of upper-case}),.)

If the difference is not the very first piece of data, the difference needs further recovering.

Apply the same processes to continuously read every pair of the output codeword and the bit string(s), if any exist. Continuously, transform them to symbol(s); then convert the symbol(s) to the absolute value of the magnitude in the matching range and then adjust it as previously instructed with the decimal value of bit string to obtain differences. The final adjustment is the decoded number added to a negative sign when an upper-case symbol is encountered; otherwise, the retrieved values remain positive or zero. In this stage, the differences of the original data have been retrieved. To obtain the sequence of the original data from the differences, simply copy the first data as the previous-data. Take the next data and set it as the current-data. Add the previous-data and the current-data; copy the result as the next decoded data. Then, the current-data becomes the previous-data; the next data of the encoded output becomes the current-data. Repeat the processes until the end of the encoded output. The copied sequence of numbers will be losslessly equal to the original data, if the processes are applied correctly.

8. Why the Design of Modified Tunstall

This project designed Modified Tunstall Coding based upon the pre-computed statistics and rewrote the compressed data in DDL. The limitations of DDL restrained one of the main design principles of Modified Tunstall Coding. This project then addressed how to make DDL efficiently represent the differences of the seismic data and gain the optimal compression ratio. The intention of this design is illustrated in this section. The

Codeword Table of Modified Tunstall Coding needs to be stored as part of the compression. This section also clarifies other design principles of Modified Tunstall Coding by giving the reasons for reducing the Codeword Table and grouping the ranges of the differences on Table 2.

In order to work with DDL, as illustrated earlier in the section “4. Data Description Language of SEED,” the design of this project requires simple computations to fit the limitation of DDL. In general, a method using less computation results in poor compression; a method using more computation, such as linear prediction which performs a good compression, may not be handled by the simple primitives of DDL. To accommodate the limitation of DDL, this project chose to modify Tunstall Coding and made it efficient to represent data. Also, there were other reasons, such as keeping the size of the Codeword Table small. These reasons for the design are illustrated in the following paragraphs.

The differences in the sequence of integers of seismographic data between consecutive samples are generally much smaller than a 32-bit integer. Typically, the differences in the sequence of integers of seismic data require 13 bits or fewer to represent them [4]. If each difference is a 13-bit integer, at the worst case, the total differences may need 2^{13} different symbols to encode them. If the reader expands the 2^{13} different symbols to be the initial alphabet size of the sequence of symbols in the Tunstall tree, then a large size Codeword Table can be expected. As introduced earlier in the Section of “5. Tunstall Coding,” the size of the Tunstall tree, defined as 2^n , has to be greater than the size of the different symbols m , where n is the desired length of output codeword; therefore, the size of the Codeword Table may approximate 2^n , where n is larger than 13.

The Codeword Table needs to be stored as part of the compression. It is anticipated that an output with a large size Codeword Table will introduce a poor compression ratio. It is critical to reduce the table size in order to perform a fair compression ratio. The output of the unmodified Tunstall comprises approximately the 2^n -size Codeword Table and the pairs (output codeword, bit strings). Obviously, if the size of the different symbols exceeds 2^{13} , then the length of the output codeword must be bigger than 13 in order to have enough codewords to represent all the input data. In this

circumstance, both the size of the Codeword Table and the encoded pairs will be increased. When n increases by 1, it means the size of the Codeword Table will be doubled, compared with its initial size; if readers were to use it, the unmodified Tunstall would be destined to a poor compression ratio.

Tunstall Coding is known as an optimal variable-to-fixed length for memoryless source. The compression rate of Tunstall is close to the entropy. Tunstall Coding assigns a set of input strings, which are of similar frequencies, to fixed-length codeword bits. Modified Tunstall Coding basically takes the idea of the fixed-length codewords bits but maps from the codeword to the bit strings to present a piece of difference data. According to the SEED Reference Manual [4], 99 percent of the absolute value of the seismic differences could be smaller than an 8-bit. Obtaining better data compression requires further analysis on the differences' frequency. How can this project distribute the 99 percent differences evenly into each range? To answer this question relates to obtaining information about how to acquire and select the seismic data.

In Section "6 Encoding Modified Tunstall Coding," the project created Table 2 to map differences to various length ranges. The length of ranges is designed as the power of two with the intention of properly taking advantage of conserving electronic data storage space. When all integers in a range have the same probability of occurring and the length of ranges is a power of two then no further compression is possible. The design's intention of Modified Tunstall Coding is to group similar but unequal frequencies of each difference occurrence together. There were 28 files (see [Appendix II](#)) chosen for this paper. (The reason for choosing these files will be described in Section 9.) When observing one of these files, the frequencies of occurrence between -127 to 127 are similar. Therefore, this project maps the initial differences in ranges (see [Table 2](#)). The ranges are from -127 to -1, 0 to 127 and then the rest of the ranges are mapped based on the size of 2, starting with the size of the range as 2^7 , 2^8 , 2^9 Typically, the differences in the sequence of integers of seismic data require 13 bits or fewer to represent them [4]. The greatest tested difference encountered in these 28 files fell in the range of 2^{15} to $2^{16} - 1$. However, this project made $(2^{18} - 1)$ as the greatest tested difference and $-(2^{18} - 1)$ as the lowest tested difference to ensure that all differences were included.

9. Compression Ratio of Modified Tunstall Coding

This section demonstrates in detail how to calculate the compression ratios of the Modified Tunstall Coding over the previously chosen files. The section also records the compression ratio experiments' results of Modified Tunstall Coding for each of the 28 files under various lengths, from 6 to 12-bit, of the fixed-length output codeword. The average of the optimal compression ratios at each specific output codeword results in 3.19.

A way to measure the efficiency of Modified Tunstall Coding is to calculate the compression ratio. The compression ratio for a particular file is the number of bits required to represent the file before compressing it to the number of bits required to represent the file after compression. The compression ratio is expressed below:

$$\text{Compression ratio} = \frac{\text{the number of bits required to represent the file before compressing}}{\text{the number of bits required to represent the file after compressing}}$$

This project requested seismic data in SEED format from IRIS's DMC center. In order to practice on the uncompressed data, SEED files have to be converted to other formats, such as ASCII or SAC, through RDSEED application. The SAC format was selected for this project.

This project assumes that each piece of seismic data sample, which is an integer, needs a 32-bit format. Consequently, the size of the file required to represent the file before compression is obtained from the total integer samples times 32 bits.

$$\text{size of file before compressing} = 32 * \text{the total integer samples required}$$

The total size required to represent the compressed file, after Modified Tunstall coding, needs to include the size of the Codeword Table and the combination of output codewords and bit strings. Note, this project supposes every alphabet symbol in the Codeword Table requires 8 bits storage space, and each output codeword size could be set in the 5- to 12-bit range. (The reason of this paper only presents the length of the codeword from 5 to 12 is illustrated in this Section in the paragraph following Table 3.)

Since the output codewords are of a fixed length, simply total the number of the output codewords and multiply the total with the fixed length bits. Measuring the size of the bit strings requires the sum of the individual length of each corresponding bit string.

Therefore, the total size of a file after compression is written as follows:

$$\text{size of file after compressing} = \text{size of Codeword Table} + \text{size of (output codewords, bit strings)}.$$

The compression ratio of Modified Tunstall coding is obtained with the following equation:

$$\text{compression ratio} = \text{size of file before compressing} / \text{size of file after compressing}$$

However, this project only concentrated on seismic data samples and removed all the data headers from each individual file. Therefore, after decompressing the files by RDSEED and running them in this project's designed application, all the SAC headers were removed, leaving only record numbers of the seismograms contained in each volume. A file containing only record numbers without headers is called seismic data, in this project. The seismic data is a sequence of integers. For practical reasons, this project only targeted the large files. The experiments were performed only on large files, after recovering from RDSEED, which are larger than 10^5 bits long. For the later experiments on the compression schemes, 28 files, without headers and greater than 10^5 bits long, were used (see [Appendix II](#)). These 28 files are numbered from 1 through 28 for later use.

Table 3. Compression Ratios of Modified Tunstall

Files #	# of samples	compression ratio at given codeword length							Max. Ratio
		6	7	8	9	10	11	12	
1	17.33	3.34	3.51	3.60	3.66	3.69	3.70	3.68	3.70
2	17.29	2.46	2.57	2.62	2.62	2.64	2.66	2.64	2.66
3	19.98	2.66	2.77	2.77	2.78	2.82	2.80	2.81	2.82
4	19.98	2.57	2.66	2.67	2.68	2.70	2.71	2.69	2.71
5	17.29	2.62	2.72	2.73	2.73	2.77	2.76	2.74	2.77
6	34.56	3.37	3.52	3.61	3.66	3.70	3.73	3.73	3.73
7	34.56	3.37	3.53	3.61	3.67	3.70	3.72	3.73	3.73
8	34.56	3.32	3.45	3.52	3.60	3.64	3.66	3.63	3.66
9	17.28	3.83	3.86	3.88	3.89	3.89	3.88	3.83	3.89
10	17.28	3.90	3.92	3.93	3.93	3.92	3.90	3.85	3.93
11	17.28	3.83	3.86	3.88	3.89	3.89	3.88	3.83	3.89
12	17.28	3.34	3.50	3.60	3.66	3.69	3.70	3.68	3.70
13	17.28	3.35	3.50	3.59	3.64	3.68	3.69	3.67	3.69
14	17.28	3.33	3.48	3.58	3.64	3.67	3.68	3.67	3.68
15	19.99	2.62	2.72	2.73	2.72	2.77	2.76	2.75	2.77
16	17.28	2.43	2.53	2.57	2.57	2.58	2.60	2.59	2.60
17	19.99	3.05	3.19	3.20	3.23	3.26	3.26	3.27	3.27
18	19.99	3.09	3.18	3.23	3.27	3.30	3.30	3.30	3.30
19	20.00	2.92	3.01	3.03	3.05	3.08	3.07	3.08	3.08
20	17.28	2.59	2.73	2.78	2.78	2.83	2.83	2.83	2.83
21	17.28	2.61	2.77	2.79	2.80	2.85	2.85	2.85	2.85
22	17.28	2.51	2.64	2.69	2.69	2.72	2.73	2.71	2.73
23	14.60	2.50	2.63	2.67	2.67	2.69	2.70	2.68	2.70
24	14.59	2.68	2.78	2.78	2.79	2.84	2.82	2.81	2.84
25	14.58	2.63	2.73	2.74	2.74	2.78	2.77	2.75	2.78
26	14.57	2.87	2.95	2.97	2.99	3.03	3.02	3.02	3.03
27	14.57	2.89	2.92	2.97	3.01	3.03	3.03	3.01	3.03
28	14.56	2.74	2.82	2.83	2.86	2.87	2.87	2.88	2.88
Average		2.98	3.09	3.13	3.15	3.18	3.18	3.17	
Average compression ratio of each max ratio									3.19

The bold numbers highlight the maximum compression ratio.

Note: See Appendix II for the list names of selected files.

The main factors that influence compression ratios of Modified Tunstall Coding include the data's inherent characteristics, the assigned length of the output codewords, and the design of the ranges. Table 3 shows the experiments' results on the various assigned lengths--from 5-bit up to 12-bit--of the output codeword. Notice Tunstall Coding is designed so that the initial alphabet size of the sequence of symbols m has to be less than 2^n , where n is the length of the targeted output codewords. Be aware that "the sequence of symbols" means the sequence of differences in this project. A large, more than 10^5 bits, seismic file normally has the initial alphabet size exceeding 2^5 . When the length of the output codewords is assigned less than 5, the experiments result in a small Codeword Table which is too small to represent all the distinct differences. When the length of the output codewords n is assigned greater than 12, the compression ratios for the chosen 28 files decreased when compared to $n = 12$; the majority of the compression ratios decreased when $n = 13, 14$. When the output codewords are increased, the Codeword Table size is increased, the compressed length of the file will increase, and then the compression ratio decreases. Therefore, this paper only presents the length of the codewords from 5 to 12.

In Table 3, the best compression ratio, according to the length of the output codewords, results in the average of 4.22 when the output codewords are assigned to 10 or 11-bit lengths. This project chose the best compression ratio of each specified file and recorded it in the last column of Table 3. The average compression ratio of each file's optimal ratio, between 6- to 12-bit output codewords, is 3.19.

How is the performance of 3.19 compression ratio? This project took the Modified Tunstall Coding results to compare with coding algorithm--the Linear Prediction Coding in the section 11 and confirmed the quantities of the improvement of Modified Coding from Steim1/Steim2 in section 10.

10. Comparing Modified Tunstall with Steim1 and Steim2

This section takes the results of Modified Tunstall Coding in the last section, individually comparing them with Steim1 and Steim2. The results of the comparisons are presented in Table 4. The average improvement of Modified Tunstall Coding is 30.78%

over Steim1 and 5.16% over Steim2. The last paragraph of this Section discusses the percentages of results.

The compression algorithms applied in DDL of SEED for seismic data are Steim1 and Steim2 compression schemes. This project has presented the new compression scheme, Modified Tunstall Coding, to improve Steim1 and Steim2. To understand how much improvement that Modified Tunstall Coding made with the other compression schemes requires comparison. One common way to compare is to measure the performances by comparing the compression ratios among the compression schemes.

To measure the compression performances, this project did not use Steim1 and Steim2 compression algorithms that are used in SEED because they are not independent applications for measuring the compression ratio. Steim1 and Steim2 compression algorithms in SEED are written in DDL language. To rewrite them into an independent application takes a tremendous effort. A convenient way to find an application that adapts Steim1 and Steim2, steim123, is collected by the Quanterra Users Group (QUG) of the Northern California Earthquake Data Center at <http://www.ncedc.org/qug/software/>. The steim123 tar file downloaded from the Web site of the QUG consists of three main classes: steim123.c, swrseed.c, and srdseed.c. Only the steim123.c is related, in this project, to compressing integer seismic data. Level 1 and Level 2 of steim123 are equivalent to Steim1 and Steim2 of the SEED format, respectively. This paper recodes Level 1 as Steim1 and Level 2 as Steim2 in the following descriptions.

In general, a seismometer's output data under normal background are usually highly-correlated integers. Steim1 and Steim2 use this highly-correlated characteristic of seismic data to compress data. By applying this significant characteristic, Steim1 and Steim2 calculate the differences of the sequence of integers, as well as of Modified Tunstall Coding. A significant characteristic of the typical differences of seismic data is that integer differences are generally much smaller than 32-bit. Accordingly, Steim1 and Steim2 assign the first-differences into groups of numbers of variable bits long. Steim1 codes the first-differences as groups of 8-, 16-, or 32-bit. Steim2 improved upon Steim1 and codes groups of differences of 4-, 5-, 6-, 8-, 10-, 15-, and 30-bit. Steim1 and Steim2 use the control codes, introduced in section "4. Data Description Language of SEED," to figure the configuration of the differences.

To measure the relative increase or decrease of the compression ratios of Modified Tunstall Coding (MTC), this project views the compression ratio of Modified Tunstall Coding as the initial value; the compression ratio of the other schemes, such as Steim1, Steim2, and Linear Prediction Coding, as the total which corresponds to 100%.

The percentage improvement in compression ratio between Modified Tunstall Coding and the other compression scheme in each specified file is calculated as:

The percentage improvement in compression ratio (%)

$$= (\text{MTC's ratio} - \text{other's ratio}) * 100 / \text{other's ratio}$$

According to Appendix B of RDSEED V2.4, the best compression ratio of Steim2 is 6.74 when all the differences can be compressed in 4 bits. The same file is only compressed in the ratio of 3.67 when using Steim1. This maximum improvement from Steim1 to Steim2 is 65.40%. The calculation is shown as follows:

$$((6.74 - 3.67) * 100 / 3.67) \% = 65.40\%$$

The average improvement from Steim1 to Steim2 of these chosen files is 26.27% as shown in Table 4. Recall: Steim2 groups 4-, 5-, 6-, 10-, 15- or 30-bits first-differences, but Steim1 encodes first-differences into 8-, 16- or 32-bit. This is the main reason that Steim2 performs better than Steim1. Another of the reasons is that the compression ratio of Steim2 was achieved from Steim1 on 20 Hz Broad Band data, but only 13 out of the 28 files are 20 Hz data (see [Appendix III](#)).

Table 4 demonstrates that Modified Tunstall Coding compresses the 28 chosen files on an average of 30.78% better than Steim1. Each individually tested file confirms that the compression ratio of Modified Tunstall Coding is the worst at 1.00% and the best at 54.22% compared with Steim1. Recall: Steim1 only uses the methods, which are the first-differences of groups 8-, 16-, or 32-bit, but Modified Tunstall Coding encodes differences into 7- to 17-bit (see Table 2). The reason that Modified Tunstall Coding demonstrates the optimistic average percentage better than Steim1 is because Modified Tunstall Coding covered all the Steim1's coding methods.

Table 4. Comparison of Steim1/Steim2 and Modified Tunstall

File #	Compression ratios			percentage improvement in compression ratios			First order Entropy bits/symbol	Max. possible compression ratios
	(1)	(2)	(3)	(4)	(5)	(6)		
	Steim1	Steim2	MTC	Steim2 over Steim1	MTC over Steim1	MTC over Steim2		
				%		%		
1	3.74	5.33	3.70	42.51	-1.00	-30.53	5.61	5.71
2	1.87	2.05	2.66	9.63	42.32	29.83	11.60	2.76
3	1.89	2.33	2.82	23.28	49.26	21.07	10.90	2.94
4	1.88	2.13	2.71	13.30	44.10	27.18	11.40	2.81
5	1.88	2.24	2.77	19.15	47.29	23.62	11.12	2.88
6	3.64	3.85	3.73	5.77	2.48	-3.11	7.72	4.15
7	3.65	3.87	3.73	6.03	2.19	-3.62	7.71	4.15
8	3.39	3.69	3.66	8.85	7.90	-0.87	8.12	3.94
9	3.74	5.51	3.89	47.33	4.05	-29.38	5.64	5.67
10	3.74	5.61	3.93	50.00	5.04	-29.97	5.52	5.80
11	3.74	5.34	3.89	42.78	4.05	-27.13	5.88	5.44
12	3.71	5.33	3.70	43.67	-0.20	-30.53	5.61	5.71
13	3.70	5.26	3.69	42.16	-0.20	-29.80	5.73	5.59
14	3.70	4.98	3.68	34.59	-0.42	-26.01	5.94	5.39
15	1.88	2.24	2.77	19.15	47.40	23.71	11.11	2.88
16	1.87	1.99	2.60	6.42	39.11	30.72	11.87	2.70
17	2.21	2.99	3.27	35.29	47.84	9.27	9.20	3.48
18	2.36	3.05	3.30	29.24	40.01	8.34	9.05	3.54
19	2.00	2.78	3.08	39.00	54.22	10.95	9.85	3.25
20	1.90	2.38	2.83	25.26	49.12	19.05	10.73	2.98
21	1.90	2.44	2.85	28.42	50.22	16.97	10.61	3.02
22	1.88	2.15	2.73	14.36	45.10	26.88	11.24	2.85
23	1.88	2.13	2.70	13.30	43.65	26.79	11.34	2.82
24	1.89	2.35	2.84	24.34	50.11	20.72	10.82	2.96
25	1.88	2.25	2.78	19.68	48.07	23.72	11.05	2.90
26	2.05	2.71	3.03	32.20	47.56	11.63	10.11	3.17
27	2.10	2.74	3.03	30.48	44.24	10.55	10.05	3.18
28	1.94	2.51	2.88	29.38	48.22	14.56	10.65	3.00
Average	2.57	3.29	3.19	26.27	30.78	5.16	9.15	3.77

- (1) The compression ratio of Steim1.
 - (2) The compression ratio of Steim2.
 - (3) The compression ratio of Modified Tunstall Coding.
 - (4) Percentage improvement in compression ratio of Steim1 over Steim2
 - (5) Percentage improvement in compression ratio of MTC over Steim1.
 - (6) Percentage improvement in compression ratio of MTC over Steim2
- Notice: the highlighted rows are DGAR or PTCN stations.

Modified Tunstall Coding compressed the 28 chosen files on average 5.16% better than Steim2 (see Table 4). When compared with Steim2, Table 4 also shows that Modified Tunstall Coding is worst at 30.53% and best at 30.72%. One of the main reasons is that Steim2 encodes groups of differences of 4-, 5- and 6-bit but Modified Tunstall Coding starts to encode small differences from 7-bit. When files include many small differences under 7-bit, the inability of Modified Tunstall Coding obviously will not have the optimistic average percentage. Recall: The frequencies of occurrence between -127 to 127 are similar. This constraint is violated in the files of Steim2's compression ratios, which are better than Modified Tunstall Coding.

The first order entropy of the sequence of integer differences between successive samples is calculated in Table 4. Assuming Modified Tunstall Coding is coded at the first order entropy ratio then the maximum possible compression ratio can be calculated (see Table 4). No Modified Tunstall Coding can improve any more than its maximum possible compression ratio.

Table 5. Comparison of Steim1, 2 and MTC at DGAR/PTCN Stations

File #	Compression ratios			percentage improvement in compression ratios			First Order Entropy bits/symbol	Max. possible compression ratios
	(1)	(2)	(3)	(4)	(5)	(6)		
	Steim1	Steim2	MTC	Steim2 over Steim1	MTC over Steim1	MTC over Steim2		
				%		%		
2	1.87	2.05	2.66	9.63	42.32	29.83	11.60	2.76
3	1.89	2.33	2.82	23.28	49.26	21.07	10.90	2.94
4	1.88	2.13	2.71	13.30	44.10	27.18	11.40	2.81
5	1.88	2.24	2.77	19.15	47.29	23.62	11.12	2.88
17	2.21	2.99	3.27	35.29	47.84	9.27	9.20	3.48
18	2.36	3.05	3.30	29.24	40.01	8.34	9.05	3.54
19	2.00	2.78	3.08	39.00	54.22	10.95	9.85	3.25
20	1.90	2.38	2.83	25.26	49.12	19.05	10.73	2.98
21	1.90	2.44	2.85	28.42	50.22	16.97	10.61	3.02
22	1.88	2.15	2.73	14.36	45.10	26.88	11.24	2.85
26	2.05	2.71	3.03	32.20	47.56	11.63	10.11	3.17
27	2.10	2.74	3.03	30.48	44.24	10.55	10.05	3.18
28	1.94	2.51	2.88	29.38	48.22	14.56	10.65	3.00
Average	1.99	2.50	2.92	25.31	46.88	17.68	10.50	3.06

An observing of the data of Table 4, shows all the compression ratios of Modified Tunstall Coding of DGAR and PTCN stations are better than Steim2; mostly, other stations are worse than Steim2. This project compiles DGAR and PTCN stations from Table 4 into Table 5. Observing the results of Table 5, Modified Tunstall Coding compresses DGAR and PTCN stations' files on average 46.88% and 17.68% better than Steim1 and Steim2, respectively.

Steim1 and Steim2 are less expensive in computation; however, they do not perform optimal compression ratios. In contrast, obtaining a better data compression scheme, in general, requires more expensive computation, such as linear prediction. It is challenging to design a compression scheme that does not require heavy computation to fit DDL requirements and limitations. Modified Tunstall Coding has suitable computation to be handled by DDL. This project applied Modified Tunstall Coding by looking at statistical differences of the seismic data. Steim1 and Steim2 simply applied a fixed method to assign the data to fewer than 32-bits, regardless of the essence of the data. The pre-computing of the statistics on profile bases made a remarkable improvement on the compression ratios of Modified Tunstall Coding over Steim1 and Steim2.

11. Comparing Modified Tunstall with Linear Prediction Coding

One of the most powerful compression schemes is Linear Prediction Coding (LPC). Linear prediction, a mathematical operation, estimates upcoming values of a discrete time signal as a linear function of previous samples. In digital signal processing, linear prediction is often called linear predictive coding. It is a tool for representing data in compressed form and uses the information as a linear predictive model.

To compare Modified Tunstall Coding with LPC, this project has taken the concept of the LPC application, Predictcode.m, from the textbook, *Digital Signal Processing with Examples in Matlab* by Sammel D. Stearns [7], and rewritten it to fit the project's needs. Predictcode.m application (see [Appendix I](#)) de-correlated data from the input file, with a small residual, by choosing a weight vector. The residual is stored in a separated file and is compressed by Arithmetic Coding [8]. Each compressed result is compared with the compression ratio of Modified Tunstall Coding.

Arithmetic Coding is commonly used in lossless compression schemes. It is a flexible statistical compression method and encodes at or close to entropy when the probabilities of symbols are skewed. Arithmetic Coding compresses each input stream into a unique interval in $[0, 1)$ while other entropy encoding schemes replace codes to each individual symbol. The method reads each input stream, symbol by symbol, and appends significant bits to the code by inputting and processing each symbol. The Arithmetic Coding [8] applied to this project is based on the article, “Arithmetic Coding Revisited” [9]. It is written in C language.

A general idea in compressing the chosen 13 files at DGAR and PTCN stations, as this project has done using Modified Tunstall, and comparing them to the prediction coding, is described as follows:

- (1). Transform seismic data by applying the predictcode.m application (see [Appendix I](#)). This process produces the weight vector b and writes the residual e to a new file. (Note: The 6 values of the weight vector b require 20 bits each; afterward, 8 bits are required per sample.)
- (2). Apply Arithmetic Coding [8] for the residual e file, and then the result will be a byte-unit file. Calculate the byte-unit file’s size in bit by multiplying by 8.
- (3). Both weight vector and the byte-unit file in bit of the residual have to be stored as compression. Therefore, the total number of bits of the compressed data is
Compressed bits = the weight vector bits + (2)’s Arithmetic Coding results
- (4). Divide the bits of the original file by the total number of bits of compressed data to form the compression ratio of the prediction coding.

$$\text{Compression ratio} = \frac{\# \text{ bits of the original file}}{\# \text{ bits of compressed data}}$$

Table 6 demonstrates the 13 chosen files of compression ratios on Linear Prediction Coding and Modified Tunstall Coding at DGAR and PTCN stations. The average compression ratio of the Modified Tunstall Coding is lower than LPC coding at 7.95%. As this paper described earlier, Arithmetic Coding encodes data at or close to entropy when the probabilities of symbols are skewed.

Table 6 also computes the sequence of differences’ first order entropies of the 13 chosen files. Recall: This project assumes that each piece of seismic data sample needs a 32-bit format. After dividing 32-bit per symbol by the first order entropy of each file, the

maximum possible compression ratios of each file are shown in Table 6. The compression ratio of LPC can potentially achieve a higher ratio than the maximum possible compression ratio of other schemes because LPC account for more sample-to-sample correlation. Due to the maximum possible compression ratio, Table 6 results also show that the Modified Tunstall can not improve any more than an average 4.65% for these 13 files; even though, Table 6 shows that Modified Tunstall Coding is worse than LPC coding at average 7.95%.

One advanced improvement for further research to improve the compression ratio is to evenly rearrange the data differences into various ranges as was discussed in section “8. Why the design of the range.” The advanced improvement remains for the readers for further research.

Table 6. Comparison of LPC and Modified Tunstall at DGAR/PTCN Station

	Compression ratios		percentage improvement in compression ratios			
	(1)	(2)	(3)			(4)
	MTC	LPC	MTC over LPC	First order Entropy	Max possible ratios	MTC over Entropy
			%	bits/symbol	%	
2	2.66	2.81	-5.29	11.60	2.76	-3.50
3	2.82	3.03	-6.90	10.90	2.94	-3.93
4	2.71	2.91	-6.91	11.40	2.81	-3.51
5	2.77	2.95	-6.13	11.12	2.88	-3.77
17	3.27	3.61	-9.50	9.20	3.48	-6.02
18	3.30	3.64	-9.22	9.05	3.54	-6.58
19	3.08	3.64	-15.26	9.85	3.25	-5.05
20	2.83	3.16	-10.34	10.73	2.98	-4.96
21	2.85	3.2	-10.81	10.61	3.02	-5.35
22	2.73	3.01	-9.37	11.24	2.85	-4.15
26	3.03	2.92	3.60	10.11	3.17	-4.46
27	3.03	3.32	-8.77	10.05	3.18	-4.87
28	2.88	3.14	-8.43	10.65	3.00	-4.29
Average	2.92	3.18	-7.95	10.50	3.06	-4.65

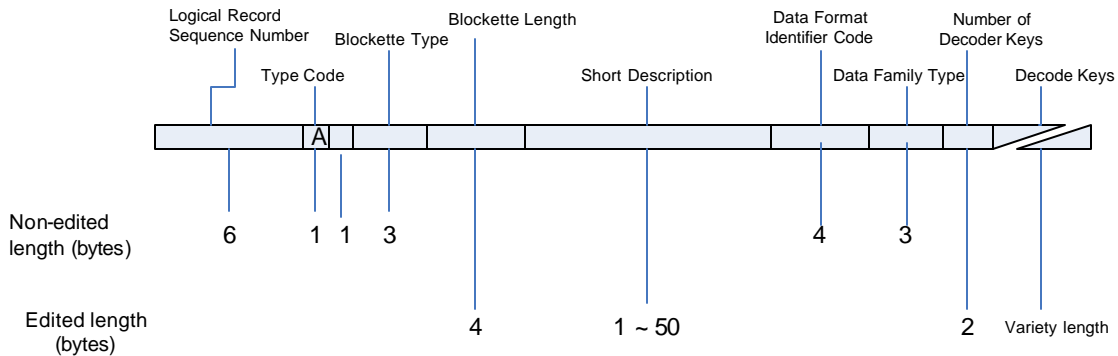
- (1) The maximum compression ratios on the specified codeword length of MTC at DGAR/PTCN stations.
- (2) The compression ratios of Linear Prediction Coding (LPC) at DGAR/PTCN stations.
- (3) Compare the compression ratios of LPC and MTC at DGAR/PTCN stations.
- (4) Compare the compression ratios of MTC and the maximum possible ratios at DGAR/PTCN stations.

12. Rewriting Modified Tunstall Coding in DDL

To ensure that Modified Tunstall Coding can be rewritten in DDL and properly run by SEED reader, the last part of this project rewrote the Modified Tunstall compression scheme in DDL. This section outlines the essential organization, the design of DDL code for the Modified Tunstall Coding system, and the reedited relative portion of DDL's SEED format. Finally, the rewritten DDL of SEED is successfully tested by the RDSEED application.

SEED format consists of four control headers: Volume Identifier Headers, Abbreviation Dictionary Control Headers (ADCH), Station Control Headers, and Time Span Control Headers. SEED also uses data recorder headers. Each control header, containing information about complete a logical volume, is coded in a computer readable hierarchy of data structures. Each header utilizes one or more data structures, which are strung in specified order. The structure contains fixed and variable data fields. Figure 4 shows the view of a data structure example of the ADCH.

Figure 4. View of the Dictionary Header



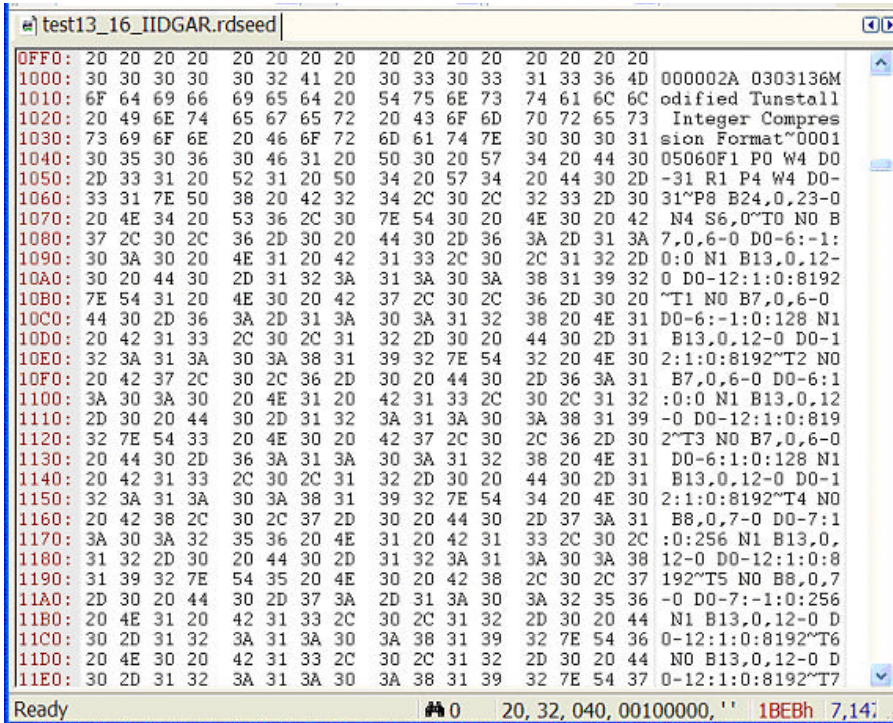
Each data structure contains an identification code, a length specification, and a sequence of related data fields. Formatted data structures are used in control headers and unformatted data structures are used in the header portions of data records. The actual

decoding command is defined in the ADCH, which abbreviates the ASCII data description string. For more information on SEED, see the reference manual of SEED format version 2.4[4].

SEED format allows for one or more volumes. This implies that a SEED file may include one or more data records. To simplify the experiments, this project tested one volume at a time on a single seismometer of a specified network station. In order to specify the parameters of the station List, Channel List, Network List, and the Location ID in the SEED reader, this project simplified each individual SEED file that spanned only one volume, on a single seismometer of a specified network station, by RDSEED. The simplified processes could create one or more SEED-formatted files, which included one volume on a single seismometer of a specified network station.

To write Modified Tunstall Coding in DDL, two portions of the SEED file, the ADCH and the Data Record, had to be reedited. The ADCH had four sections that required editing. Figure 4 depicts a view of the edited and non-edited sections of the Dictionary Header. The ADCH is written as an alphanumeric ASCII string, which is recorded as “A” Type Code in the Header as Figure 4. This project searched the control header at the 7th byte that has “A” Type Code of the header, and then rewrote its Blockette Length, according to the actual length of the entire control header without counting the first eight bytes, at the 12th to the 15th byte. The Short Description, which is a short name describing the compression format applied in this file, needed to be rewritten starting at the 16th byte with the 50 bytes maximum length. This project placed a tildes, ~, at the last byte of the Short Description indicating the end of the description. This project copied the Data Format Identifier Code and the Data Family Type, which occupied a total of 7 bytes, without any change. Then it modified the Number of Decoder Keys, 2 bytes long, according to the actual number of decoder keys. Finally, the project wrote all the Decoder Keys in the sequence, one-by-one, separating each key by using a tilde.

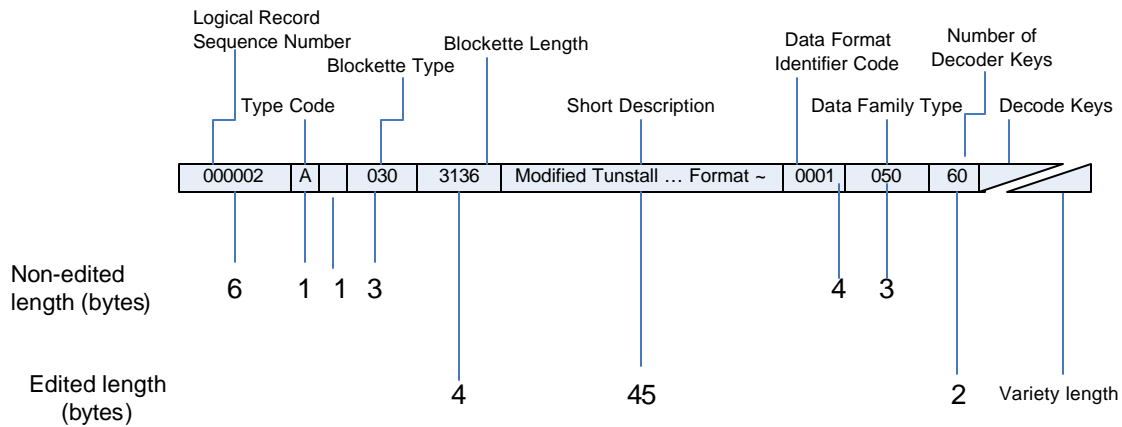
Figure 5. Snapshot of an Example of the Dictionary Header



To ensure that the readers are able to write the DDL code for modified Tunstall, a snapshot of an actual example of the ADCH is given in Figure 5. The ADCH has a 4096 bytes length. It starts on the 1000 line at the left hand column of Figure 5 and ends at the 1FF0 line. The details of the initial 69 bytes of the ADH are demonstrated in Figure 6. Overall, the data family type 050 uses 60 keys, which are located at the “Number of Decode Keys” of Figure 5. Family type 50 is the Integer Differences Compression. The 60 keys include the first 2 non-control type keys and 58 of control type keys. The first key, key1: *F1 P0 W4 D0-31 R1 P4 W4 D0-31*, is separated by a tilde with the second key. Key 1 describes the very first constant *F1* of the sequence of differences that is located at the 0th byte (*P0*) of the ADCH, and then 4 bytes are copied (*W4*), extracted bits 0-31 (*D0-31*), from the input stream into the working buffer; the last constant (*R1*) of the sequence of differences is located at the 4th byte (*P4*) of the ADCH, and then 4 bytes are copied (*W4*), extracted bits 0-31 (*D0-31*), from the input stream into the working buffer. Key 2, *P8 B24, 0, 23-0 N4 S6, 0*, illustrates that 24 bits are copied (*B24, 0, 23-0*) from the

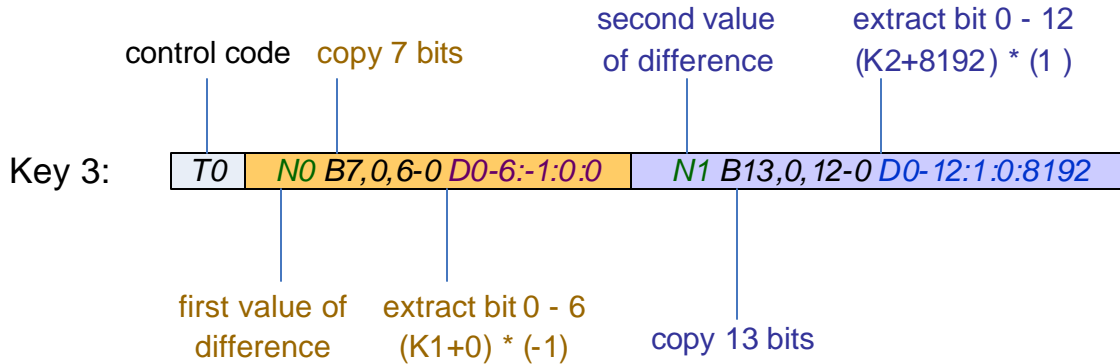
input stream, starting from the first bit of the 8th byte (*P8*) to the last bit (23-0) (crossing byte boundaries if necessary); the 24 bits include 4 control codes (*N4*), which are 6-bit wide (*S6*). Note: The bits are numbered in big-endian.

Figure 6. View of the ADCH of Figure 5



Control type keys, key 3 to key 60, carry out the action required by the control code derived in key 2. Key 3 to key 60, are equivalent to the stored Codeword Table of Modified Tunstall Coding. For example, because key 3 to key 60 have similar interpretations, this paper will take key 3 (see Figure 7) and interpret it in a more understandable text. Key 3, *T0 N0 B7, 0, 6-0 D0-6: -1: 0: 0 N1 B13, 0, 12-0 D0-12:1: 0: 8192*, maps the “Ah” input string (see Table 2) to its corresponding ranges. *T0* describes how to decode data when the control code with the value 0 is encountered. To decode the first value of difference *N0*, copy 7 bits, starting from the first bit to the 0th bit (*B7, 0, 6-0*), and extract bits 0-6 to form unsigned integer value k_1 and then apply offset 0 and scale factors -1 as specified by code 0 (*D0-6: -1: 0: 0*). To decode the second difference value *N1*, copy 13 bits, start from the first bit to the 6th bit (*B13, 0, 12-0*) and extract bits 0-12 to form unsigned integer value k_2 and then apply offset 8192 and scale factors 1 as specified by code 0 (*D0-12:1: 0: 8192*).

Figure 7. View of key 3



As this paper mentioned in section “8. Why the Design of Modified Tunstall,” the Codeword Table and the pair of output codewords and bit strings need to be stored as Modified Tunstall Coding. The Codeword Table is stored as the ADCH in DDL. The pair of the output codewords and bit strings is stored in the Data Record portion of SEED. The Data Record section can be identified by the Data Header/Quality Indicator. The Data Header/Quality Indicator, which is located at the 7th byte of the Data Record’s structure by the upper case letters “D,” “R,” or “Q,” is called Type Code in the ADCH. The Data Record portion has a fixed length of 48 bytes. To edit the output codewords and the bit strings, the data from the 49th byte until the end of the data is reedited by attaching the compressed data of the Modified Tunstall Coding. At this point, the writing for the Modified Tunstall is finished and can be read in RDSEED.

To give an idea to SEED users on how much improvement Modified Tunstall Coding in SEED format, Table 7 demonstrates SEED compression ratio on two files. File # 2, *2005.365.23.59.07.9980.IU.PTCN.00.BHE*, is encoded in Steim2 and file #17, *2006.001.00.00.02.5999.II.DGAR.10.BHE*, is encoded in Steim1. Originally, they had a compression ratio at 1.97 and 2.16, respectively. File #2 and #17 were rewritten in MTC-SEED format and compression ratios at 2.42 and 2.99, respectively, were obtained. The improved compression ratio of MTC-SEED format is 22.84% over Steim2-SEED and 38.43% over Steim1-SEED. SEED has worse compression ratios in Table 7 compared

with Steim1, Steim2 and Modified Tunstall Coding in Table 3, 4, and 5. Two main reasons explain the low ratios of SEED format: (1) the headers of SEED are not compressed at all. (2) SEED format has fixed-reserved space in terms of Block, even though the whole Block is not filled.

Table 7. SEED Compression Ratios of Steim and MTC

File #	Original Compression Scheme	SEED Compression Ratios		SEED improvement
		Steim-SEED	MTC-SEED	MTC over Steim %
2	Steim2	1.97	2.42	22.84
17	Steim1	2.16	2.99	38.43

13. Conclusion

Over all in this project, Modified Tunstall Coding showed better compression ratios compared to Steim1 and Steim2. The comparison of Modified Tunstall Coding and Linear Prediction Coding demonstrates that Modified Tunstall Coding compressing seismic data in this project is 7.95% worse than Linear Prediction Coding for the stations of DGAR and PTCN. Modified Tunstall Coding can not improve any more than an average of 4.65% for DGAR and PTCN stations due to the result of the maximum possible compression ratio.

The interesting and challenging part of this project has been that it project crosses two science fields, seismology and data compression, and demonstrates how data compression supports the field of the seismology. Specifically, challenges, such as lack of DDL documentation and bugs existing in DDL of SEED format, were encountered and solved in the process of writing Modified Tunstall Coding in DDL. An exciting part is, in conclusion, that this project demonstrated that Modified Tunstall Coding is close to the optimal coding. Further research, which could evenly rearrange the data differences into various ranges to approach the optimal compression ratios, remains for others to explore.

14. References

- [1]. D. Salomon, Data compression, New York: Springer, 2004.
- [2]. K. Sayood, Introduction to data compression, CA: Morgan Kaufmann, 2000.
- [3]. "IRIS Data Management Center," n.d.; <http://www.iris.edu/about/DMC/>
- [4]. "Standard for the Exchange of Earthquake SEED format Version 2.4 March, 2006," n.d.; http://www.iris.edu/manuals/acrobat/T_SEED.pdf
- [5]. "rdseed Manual Pages," http://www.iris.edu/manuals/acrobat/T_rdseed.pdf
- [6]. S. A. Savari, "Variable-to-fixed length codes and plurally parsable dictionaries," *Proc. Data Compression Conference*, (1999) 453-462.
- [7]. S. Stearns, Digital Signal Processing with Examples in Matlab, CRC Press, 2003.
- [8]. J. Carpinelli, W. Salamonsen, L. Stuiver and Radford Neal, "ArithmeticCoder," <http://mg4j.dsi.unimi.it/docs/it/unimi/dsi/mg4j/io/ArithmeticCoder.html>; accessed April 1, 2006.
- [9]. A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, 16, 3, Jul. 1998, pp. 256-294, Portal Database, <http://doi.acm.org/10.1145/290159.290162>; accessed April 1, 2006.

Appendix I: Application of Predictcode.m

```
%predictcode.m is a Matlab file.  
%Took the concept from the Digital Signal Processing with Examples in Matlab By  
%Samuel D. Stearns
```

```
d=textread('2006.001.00.00.00.0479.IU.ANMO.00.BH2.Q.SAC_ASC', '%d');
```

```
K = length(d);  
N = 6;  
f = [0; d(1:K-1)];  
rff = autocovar_mat(f,N);  
rfd =crosscovar(f,d,N);  
b= rff \ rfd;  
e = d - (filter(b,1,f));
```

```
fw = fopen( '0479_recovery_round_int.txt', 'w' );  
fprintf( fw, ' %d \n', recovery);  
fclose(fw);
```

```
d=textread('2006.001.00.00.00.0104.IU.ANMO.10.BH1.Q.SAC_ASC', '%d');
```

```
K=length(d);  
N= 6;  
f=[0; d(1:K-1)];  
Rff=autocovar_mat(f,N);  
rfd=crosscovar(f,d,N);  
b=Rff\rfd;
```

```
e=[d; 0]-[0; filter(b,1,f)];  
e_1 = e(1:end-1);
```

```
fw = fopen( '0104BH1_0.txt', 'w' );  
fprintf( fw, ' %6.0f \n', e_1);  
fclose(fw);
```

Appendix II: The List of Selected File Names

The WebRequest form for requesting data from the DMC of IRIS at <http://www.iris.edu/data/WebRequest.htm>

use '?' when wildcarding	CANNOT wildcard network	STARTING TIME						ENDING TIME						use '?' when wildcarding	LOCATION
Station	Network	Year YYYY	Month MM	Day DD	Hour 00	Min 00	Sec 00.0	Year YYYY	Month MM	Day DD	Hour 00	Min. 00	Sec. 00.0	Channel names	LOC ID
DGAR	II	2006	01	01	00	00	00	2006	01	02	00	00	00	BH?	
PTCN	II	2006	01	01	00	00	00	2006	01	02	00	00	00	BH?	
ANMO	IU	2006	01	01	00	00	00	2006	01	02	00	00	00	BH?	

Appendix III: Seismic Data Request Form

File #	Selected Files	Network	Station	Location	Channel	Sample Rate
1	2005.365.23.55.41.7981.IU.MAJO.00.BHE	IU	MAJO	00	BHE	20 Hz
2	2005.365.23.59.07.9980.IU.PTCN.00.BHE	IU	PTCN	00	BHE	20 Hz
3	2005.365.23.59.17.6855.IU.PTCN.01.BHZ	IU	PTCN	01	BHZ	40 Hz
4	2005.365.23.59.21.2105.IU.PTCN.01.BHN	IU	PTCN	01	BHN	40 Hz
5	2005.365.23.59.22.8480.IU.PTCN.00.BHZ	IU	PTCN	00	BHZ	20 Hz
6	2006.001.00.00.00.0104.IU.ANMO.10.BH1	IU	ANMO	10	BH1	40 Hz
7	2006.001.00.00.00.0104.IU.ANMO.10.BH2	IU	ANMO	10	BH2	40 Hz
8	2006.001.00.00.00.0104.IU.ANMO.10.BHZ	IU	ANMO	10	BHZ	40 Hz
9	2006.001.00.00.00.0479.IU.ANMO.00.BH1	IU	ANMO	00	BH1	20 Hz
10	2006.001.00.00.00.0479.IU.ANMO.00.BH2	IU	ANMO	00	BH2	20 Hz
11	2006.001.00.00.00.0479.IU.ANMO.00.BHZ	IU	ANMO	00	BHZ	20 Hz
12	2006.001.00.00.00.0481.IU.MAJO.00.BHE	IU	MAJO	00	BHE	20 Hz
13	2006.001.00.00.00.0481.IU.MAJO.00.BHN	IU	MAJO	00	BHN	20 Hz
14	2006.001.00.00.00.0481.IU.MAJO.00.BHZ	IU	MAJO	00	BHZ	20 Hz
15	2006.001.00.00.00.7855.IU.PTCN.01.BHE	IU	PTCN	01	BHE	40 Hz
16	2006.001.00.00.00.9480.IU.PTCN.00.BHN	IU	PTCN	00	BHN	20 Hz
17	2006.001.00.00.02.5999.II.DGAR.10.BHE	II	DGAR	10	BHE	40 Hz
18	2006.001.00.00.02.5999.II.DGAR.10.BHN	II	DGAR	10	BHN	40 Hz
19	2006.001.00.00.02.5999.II.DGAR.10.BHZ	II	DGAR	10	BHZ	40 Hz
20	2006.001.00.00.03.5000.II.DGAR.00.BHE	II	DGAR	00	BHE	20 Hz
21	2006.001.00.00.03.5000.II.DGAR.00.BHN	II	DGAR	00	BHN	20 Hz
22	2006.001.00.00.03.5000.II.DGAR.00.BHZ	II	DGAR	00	BHZ	20 Hz
23	2006.001.13.51.49.2104.IU.PTCN.01.BHN	IU	PTCN	01	BHN	40 Hz
24	2006.001.13.51.56.6354.IU.PTCN.01.BHZ	IU	PTCN	01	BHZ	40 Hz
25	2006.001.13.52.42.3604.IU.PTCN.01.BHE	IU	PTCN	01	BHE	40 Hz
26	2006.001.13.52.55.9499.II.DGAR.10.BHE	II	DGAR	10	BHE	40 Hz
27	2006.001.13.53.06.4499.II.DGAR.10.BHN	II	DGAR	10	BHN	40 Hz
28	2006.001.13.53.17.6499.II.DGAR.10.BHZ	II	DGAR	10	BHZ	40 Hz